

Rappels

- Chapitre de cours: [Schémas XML](#)
- Le [TD4](#) sur les schémas (avec des exemples de schémas)
- `xmllint --noout --schema monschema.xsd monfichier.xml`

Exercice 1

1. Donnez un document XML qui correspond au schéma suivant:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleType name="Color">
    <xsd:restriction base="xsd:token">
      <!-- Couleur, en hexadecimal RGB -->
      <xsd:pattern value="[0-9A-Za-z]{6}" />
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="Coordinate">
    <xsd:sequence>
      <xsd:element name="x" type="xsd:double" />
      <xsd:element name="y" type="xsd:double" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="polygon">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="color" minOccurs="0" maxOccurs="1" type="Color" />
        <xsd:element name="point" type="Coordinate" minOccurs="3" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Corrigé: [XML](#)

2. Donnez un document XML qui correspond au schéma suivant (comme avant, mais plus étoffé):

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleType name="Color">
    <xsd:restriction base="xsd:token">
      <!-- Couleur, en hexadecimal RGB -->
      <xsd:pattern value="[0-9A-Za-z]{6}" />
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="AlphaColor">
    <xsd:simpleContent>
      <xsd:extension base="Color">
        <xsd:attribute name="transparency" use="required">
          <xsd:simpleType>
            <xsd:restriction base="xsd:float">
              <xsd:minInclusive value="0.0" />
              <xsd:maxInclusive value="1.0" />
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:attribute>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>

  <xsd:complexType name="Coordinate">
    <xsd:sequence>
      <xsd:element name="x" type="xsd:double" />
      <xsd:element name="y" type="xsd:double" />
    </xsd:sequence>
  </xsd:complexType>
```

```

<xsd:complexType name="Polygon">
  <xsd:sequence>
    <xsd:element name="color" minOccurs="0" maxOccurs="1" type="AlphaColor"/>
    <xsd:element name="point" type="Coordinate" minOccurs="3" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="NamedPolygon">
  <xsd:complexContent>
    <xsd:extension base="Polygon">
      <xsd:sequence>
        <xsd:element name="name" type="xsd:NMTOKEN"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="drawing">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="point" type="Coordinate" minOccurs="1" maxOccurs="unbounded"/>
      <xsd:element name="polygon" type="NamedPolygon" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>

```

Corrigé: [XML](#)

Exercice 2

1. Donnez un schéma pour lequel le document XML suivant est valide.

```

<programme>
  <serie statut = "fini">
    <nom>Dexter
    </nom>
    <saisons>8
    </saisons>
    <annee>2006
    </annee>
  </serie>
  <film>
    <titre>Autant en emporte le vent
    </titre>
    <annee>1950
    </annee>
  </film>
  <documentaire>
    <nom>Apocalypse
    </nom>
    <annee>2009
    </annee>
  </documentaire>
</programme>

```

2. Si nécessaire, modifiez votre schéma pour que:

1. Un `programme` accepte autant de series, films et documentaires qu'on veut (même zéro), **dans n'importe quel ordre**
2. Les sous-éléments `nom`, `annee`, `titre` et `saisons` doivent apparaître une et une seule fois dans leurs parents respectifs, et **dans n'importe quel ordre**.
: cela interdit l'utilisation des [groupes](#) et des [extensions de type](#) pour définir un "tronc commun" (`nom`, `annee`).
3. Les années doivent être entre 1880 et 2050 (inclus) (cf [Cours](#)).
Note: il vaut mieux définir un `xs:date` et le réutiliser pour ne pas le recopier à chaque fois
4. Le nombre de saisons doit être strictement positif.
5. L'attribut `statut` d'une `serie` soit obligatoire, ne puisse prendre que les valeurs `"fini"` et `"en cours"`.

6. Les `nom` et `titre` doivent commencer par une lettre majuscule. (cf [Cours](#))

Note: définissez un `pour le réutiliser avec ces deux éléments`

Note: le `type simple token` peut être utile, car dans l'exemple les nom et titre sont multi-lignes.

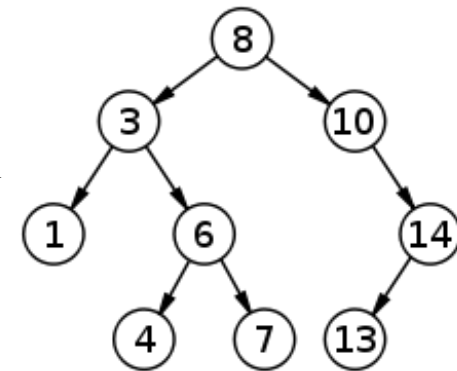
3. Vérifiez votre schéma et corrigez-le:

```
wget http://fabien.viger.free.fr/xml/td5/td5.2.tar.gz
tar -zxf td5.2.tar.gz
./td5.2.test.sh votre_schema.xsd
```

Corrigé: [XSD](#)

Exercice 3

Dans cet exercice, on va modéliser un arbre binaire de noeuds de 2 façons. Conceptuellement, un noeud contient des données arbitraires (eg. `#PCDATA` ou `CDATA`, ou `xsd:string` avec les schémas), et a un *fil gauche* et un *fil droit*, **facultatifs**, qui sont eux-mêmes des noeuds ou des références vers des noeuds. Le caractère '*gauche*' ou '*droit*' des fils est important. L'exemple ci-contre, qui est un [arbre binaire de recherche](#), le montre bien.



1. Représentation ambiguë (donc ratée)

Pourquoi ne peut-on pas *bien* représenter un arbre binaire en utilisant un seul type d'élément `noeud`?

Si vous ne trouvez pas: tentez de représenter l'exemple ci-contre. Ou est l'ambiguïté?

2. Représentation récursive

On va utiliser deux types d'éléments `noeud` et `nil`. Le premier sera l'élément racine et s'auto-inclura; `nil` servira à matérialiser l'absence de fils (gauche, droit, ou des deux).

Les données des noeuds seront dans un attribut `data` (on aurait aussi pu utiliser un contenu mixte, mais il faut choisir une convention pour avoir des corrigés).

1. Écrire l'arbre illustré ci-contre en XML. [\[Corrigé\]](#)
2. Écrire une DTD qui valide ce XML. [\[Corrigé\]](#)
3. Écrire un schéma qui le valide. [\[Corrigé\]](#)
4. Vérifiez votre DTD et votre schéma, et corrigez les:

```
wget http://fabien.viger.free.fr/xml/td5/td5.3.2.tar.gz
tar -zxf td5.3.2.tar.gz
./td5.3.2.test.sh votre_dtd.dtd votre_schema.xsd
```

3. Représentation plate et par référence

On va utiliser un élément racine `arbre` qui contiendra une liste (potentiellement vide) de `noeud`. Chaque noeud fera référence à ses fils droit et gauche (optionels) à l'aide d'**attributs**, et aura ses données propres en tant que **contenu simple**. Pensez à `ID` et `IDREF` pour les DTD, cf le [TD 3](#) ou le [Cours](#). L'élément racine devra également pointer vers le noeud qui est la racine de l'arbre.

1. Écrire l'arbre illustré ci-contre en XML. [\[Corrigé\]](#)
2. Écrire une DTD qui valide ce XML. [\[Corrigé\]](#)
3. Écrire un schéma qui le valide.
4. Vérifiez votre DTD et votre schéma, et corrigez-les:

```
wget http://fabien.viger.free.fr/xml/td5/td5.3.3.tar.gz
tar -zxvf td5.3.3.tar.gz
./td5.3.3.test.sh votre_dtd.dtd votre_schema.xsd
```

5. (*) Modifiez (si nécessaire) votre schéma pour utiliser `xsd:key` et `xsk:keyref`, cf [Cours](#). **Vérifiez** (avec la même commande).
[[Corrigé](#)]

4. Quels sont les avantages et inconvénients de ces 2 representations?
- Pensez à des arbres immenses (plusieurs millions de noeuds).
 - Pensez à la capacité de validation -- peut-on faire un arbre invalide qui passe le schéma?